1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE 1990	3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE Maintaining Incremental Optimality When Building Shortest Euclidean Tours	5. FUNDING NUMBERS
6. AUTHOR(S) Cronin, T.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army CECOM Signals Warfare Directorate Vint Hill Farms Station Warrenton, VA 22186-5100	8. PERFORMING ORGANIZATION REPORT NUMBER 92-TRF-0042
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT Statement A: Approved for public release; distribution unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Reported upon are experimental runs of an algorithm designed to incremental optimality when building tours for the Euclidean traveling salesman problem. Unlike the Lin-Kernighan edge eschange or Padberg-Rinaldi branch-and-cut techniques which begin with suboptimal tours and proceed by iterating in an attempt to converge upon or exceed the Held-Karp lower bound, the new algorithm strives to maintain optimality as each city is inserted. In previous Army research at the CECOM Center for Signals Warfare, proofs were obtained to show that the underlying search space for the Euclidean traveling salesman problem is piecemeal quartic and hyperbolic. To exploit this new knowledge, the author has developed dynamic programming algroithm which begins with a baseline tour consisting of the outer (inner) convex hull of cities, and proceeds by adding a city at a time to the interior (exterior). How the city is inserted into the existing tour is dictated by a set of quartic and hyperbolic loci which separate existing and hypothesized subtours from each other. The insertion may involve three different nonlinear operations: hyperbolic extension, quartic shunting and quartic interchange.

14. SUBJECT TERMS Artificial Intelligence, Data Fusion, Dynamic Programming	15. NUMBER OF PAGES 17
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

2

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to **stay within the lines to meet optical scanning requirements.**

Block 1. Agency Use Only (Leave blank).

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C -	Contract	**PR** -	Project
G -	Grant	**TA** -	Task
PE -	Program Element	**WU**-	Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD -	See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE -	See authorities.
NASA -	See Handbook NHB 2200.2.
NTIS -	Leave blank.

Block 12b. Distribution Code.

DOD -	DOD - Leave blank.
DOE -	DOE - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA -	NASA - Leave blank.
NTIS -	NTIS - Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

Maintaining Incremental Optimality when Building Shortest Euclidean Tours

T.M. Cronin
CECOM Center for Signals Warfare
Warrenton VA 22186-5100

Abstract.

Reported upon are experimental runs of an algorithm designed to maintain incremental optimality when building tours for the Euclidean traveling salesman problem. Unlike the Lin-Kernighan edge exchange or Padberg-Rinaldi branch-and-cut techniques which begin with suboptimal tours and proceed by iterating in an attempt to converge upon or exceed the Held-Karp lower bound, the new algorithm strives to maintain optimality as each city is inserted. In previous Army research at the CECOM Center for Signals Warfare, proofs were obtained to show that the underlying search space for the Euclidean traveling salesman problem is piecemeal quartic and hyperbolic. To exploit this new knowledge, the author has developed a dynamic programming algorithm which begins with a baseline tour consisting of the outer (inner) convex hull of cities, and proceeds by adding a city at a time to the interior (exterior). How the city is inserted into the existing tour is dictated by a set of quartic and hyperbolic loci which separate existing and hypothesized subtours from each other. The insertion may involve three different non-linear operations: hyperbolic extension, quartic shunting, and quartic interchange. To test the efficacy of these operators with regard to type 1 and 2 statistical errors, the algorithm as currently implemented is run against a benchmark of city databases for which the optimal tours are known. For those runs which result in a suboptimal solution, an explanation is sought to facilitate fixes to the formal design specification, and the code is subsequently changed. In this paper, the most recent set of runs is analyzed and reported upon, and a prognosis for scaling up to large databases is forecast. The theory predicts that a run should consume time as a function of n^3, where n is the number of cities; this bound is checked empirically by plotting city size vs. CPU time for several databases.

Background.

The Euclidean traveling salesman problem [ETSP] is a long-standing problem in optimization, having roots and primary development in the field of operations research, with ancillary developments in the fields of computational geometry and graph theory. As is the case with many obtuse problems in mathematics, the ETSP may be succinctly stated. Given a set of cities and the distances between each pair, find the shortest tour which visits each city exactly once, except the start city, which is revisited at tour's end. A tour is simply a closed loop connecting all the cities; the formal mathematical name for a tour is a Hamiltonian cycle. One of the interesting facts discovered early on is that a tour is not permitted to cross itself [F1]. There are (n-1)! / 2 possible tours through n cities, which is a combinatorially prohibitive number of operations to perform by brute force, so it is therefore desirable to find an algorithm which arrives at a solution in polynomial time. The ETSP is a special case of the general traveling salesman problem, the former bearing the distinction that the metrics involved are Euclidean distances rather than arbitrary costs or weights.

To date, the Euclidean traveling salesman problem remains unsolved. By "unsolved", it is meant that no one has developed a formal proof of optimality for a polynomial-time algorithm guaranteed to produce the shortest tour. In the mid-seventies, it was proven that the ETSP is NP-hard [G1]. This is a somewhat less damning complexity result than that obtained for the general traveling salesman problem, which belongs to the NP-complete class of problems [G2]. There have been two camps of researchers working on the Euclidean version of the problem, with the earliest computational work dating back to the end of the second world war [L2]. The first camp has striven to produce an exact solution to the problem, and in doing so has pioneered advances in the field of linear programming, including such techniques as the simplex algorithm, branch-and-bound, and branch-and-cut [P1]. An exact approach favors precision at the cost of performance. The second camp of researchers has settled for an approximate approach, by resorting to heuristics which produce high quality solutions per unit of processing time. The principal heuristic techniques are k-opt edge exchange (the most advanced of which is the iterated Lin-Kernighan), simulated annealing, genetic algorithms, elastic bands, and neural nets [J1]. Generally, the approximate techniques develop a solution with more speed than exact approaches, at the cost of precision. However, even this generality is moot, because some of the heuristic approaches render solutions orders of magnitude faster than others, with only marginally inferior solutions.

Applications.

There are a myriad of applications for the traveling salesman problem. Among them are job scheduling, resource-constrained scheduling, optimal component placement, minimal hookup wire, lowest transmission power, and path-constrained network flow [L2, J1]. The structural similarities between the problems may be somewhat subtle. For example, to map the traveling salesman problem onto job scheduling, the names of the cities are replaced by job names, and the costs between cities are replaced by the job setup times between respective jobs. The job times themselves are considered to be constants; the setup times between jobs turn out to be the crucial factor.

Some of the applications have been shown to be NP-complete problems (e.g., resource-constrained scheduling and path-constrained network flow), and therefore have failed to yield to polynomial-time algorithms [G2]. Thus, it would seem preferable at this point in time to approach such problems with approximate techniques. However, in the long term, if a polynomial-time exact solution may be obtained for the Euclidean traveling salesman problem, then it may be feasible to map the resultant algorithm onto one of the harder problems in such a way that a high quality (albeit suboptimal) solution is achieved. This mapping is arguably most suitable for that class of problems for which the triangle inequality is valid.

Verifying the Optimality of a Tour.

To test a ETSP algorithm (whether it be exact or approximate) against large databases, it is necessary to have at hand some technique to verify an optimal solution in polynomial time. For city databases of [illegible] hundred or less, it is possible to use a variant of branch-and-bound to check optimality in reasonable [illegible] [J1]. However, when n becomes much larger than one hundred, certifying optimality begins to consume unreasonable amounts of time. It is for this reason that a technique based on computing a lower bound on optimal tour length has been developed [H1]. This quantity, known as the Held-Karp lower bound, is computable in polynomial time, and empirical results indicate that it is consistently within two percent of optimal [J1]. Scientists in the field of operations research have made good use of the bound. Rather than strive for an optimal tour. researchers instead attempt to come within a reasonable neighborhood of the Held-Karp bound.

The Discovery of the Non-linear Search Space for the ETSP.

Despite over forty years of intense study by computer scientists and operations research analysts, the search space for the Euclidean traveling salesman problem remained unspecified as of 1990 (i.e., it was not known whether the mathematics of tour construction was linear, non-linear, or transcendental in the number of cities). This lack of knowledge prompted the author to conduct experiments during the winter of 1990, in an attempt to characterize the space by leveraging the recently developed field of computational geometry upon the problem. In 1968, researchers at the Johns Hopkins University reported upon a slight modification to a theorem due to Barachet to show that an optimal tour must preserve the order of the convex hull of cities - the shortest tour must contain these cities in the order in which they appear about the perimeter [B1, B2]. This fact suggested that an experiment which inserts an arbitrary city into a hull could serve as a valuable testbed in which to discover the geometric locus of equal hull perturbation. A *perturbation* is a subtour which leads into the interior of the hull through two adjacent hull vertices, to capture cities which do not lie on the hull. In conjunction with a perturbation we introduce the *elliptic distance* between a segment and a point p, which is defined to be the sum of the distances from the endpoints of the segment to p, minus the length of the segment.

When comparing a perturbed hull segment against another perturbed segment, one is actually comparing a confocal system of ellipses against another, under a continuous spectrum of elliptic distances. The foci of the two families of ellipses are respectively the two endpoints of the hull segments being perturbed. In Army research at the CECOM Center for Sig[illegible]arfare performed during the 1990 fiscal year, it was discovered that the search space induced by the intersect[illegible] the two confocal systems of ellipses is in general fourth order (quartic), and in special cases hyperbolic [C2]. [illegible]ame non-linear behavior is manifested as more cities are added to the interior, which means that the general search space is piecemeal fourth and second order regardless of the number of cities added to the tour from within the hull. Dynamic programming immediately suggested itself as an approach to the problem which might provide the framework to keep track of the quartic and hyperbolic boundaries of equal tour perturbation when a new city is added to the existing space. Armed with the the new information about the non-linear search space, the author has proceeded to develop a dynamic programming algorithm to maintain incremental optimality when building shortest Euclidean tours.

A Dynamic Programming Algorithm for the ETSP.

The algorithm is based on a principle of incremental optimality: the shortest tour containing k cities is a quartic and hyperbolic function of the shortest tour containing k-1 cities. Beginning with a baseline tour consisting of the convex hull of cities (the smallest bounding polygon containing all of the cities), one city at a time is inserted into the tour in an attempt to preserve the original optimality guaranteed by the hull. As currently specified by the algorithm, there are three types of topologies to maintain in parallel when developing a tour. First, the tour may be simply extended by inserting the new city into the space between those two cities for which the elliptic distance is smallest; this topology is termed *extension space*. A second topology is one in which the new city causes a shunt to be formed between two existing perturbations, to form a new perturbation between the two older ones; this structure is called *shunt space*. The final topology is one which deals with interchanges between perturbations which are "across the hull" from each other; this structure is termed *interchange space*. Extension space is designed to capture the hyperbolic discriminator inherent to extending an existing perturbation, whereas shunt and interchange space are models of the quartic discriminator instructing when to perform a global merger of perturbations.

A nested hull decomposition is computed during a preprocessing step. The decomposition may be computed in O [n * log n] time, as proven by Chazelle [C1]. The nested hull structure, also known as the "onion", is devised to control the order in which the interior cities are inserted. To limit the generation of greedy perturbations, those cities nearest the outer hull are installed first. The set of hulls is visited one at a time, and each hull is traversed in a counterclockwise fashion, until the set of all interior cities is exhausted. Therefore the order of insertion is dictated by a major key equal to the ordinal number of the hull in which a city resides, with a minor key equal to the relative counterclockwise position within the hull (N.B., there are exceptions based on the angle which the city forms with the tour). An alternative strategy is to begin with the innermost hull (the core of the onion) and probe outwards one hull at a time until all exterior cities are processed. Since the quartic and hyperbolic boundaries extend both inside and outside the boundary defined by the current tour, the theory guarantees that *it is legitimate to* process the nested hull decomposition in either direction, with the same optimal solution produced regardless of the processing order. An example of bi-directional processing is demonstrated in the appendix for the capitals of the forty-eight contiguous states of America.

The City Databases.

Seven sets of data (Fig. 1) are currently being used as a testbed for the dynamic programming algorithm. The first is a ten city problem published by Barachet in 1957 [B1]. The optimal tour for this small problem is discussed and derived below. The second set is a sixteen city problem which appears in a seminal computational geometry textbook [P2]. The third, fourth, and fifth sets are databases of twenty, thirty-seven, and forty-one cities which were generated to exhibit non-random behavior; they respectively represent a hull containing a single loop of interior cities; a block letter "E"; and a block letter "S". For these three databases, the shortest tours are not known with certainty (insufficient resources precluded certifying optimality with the branch-and-bound technique utilized by the operations research community), but it is conjectured that they consist of the visually-obvious structured boundaries of the hand-crafted figures. The loop dataset is discussed below, and a temporal history of the conjectured optimal tour is contained in the appendix. The sixth dataset is a forty-eight city problem solved to optimality by AT&T Bell Laboratories in 1985 [A1]. The development of its optimal solution is also contained in the appendix; both an inside-out and outside-in nested hull traversal are graphically portrayed, with the same optimal solution being obtained. The seventh and last dataset is a one hundred twenty-seven city problem formulated by the University of Augsburg in 1989 [R1]; this dataset has recently been solved to optimality by the new algorithm, but a detailed description of the optimal tour is not included here, since it will serve as a primary example in the development of a theorem to be published in a forthcoming paper [C3].

Also described below is a set of experiments in which eighty sets of cities are generated at random to be used as databases to test the analytically-derived time complexity bound for the dynamic programming algorithm.

For these eighty databases, the optimal tour lengths are unknown but are not required, for the sole purpose in using the cities is to statistically test the run time performance of the current implementation of the algorithm, independent of the fact that the solutions developed may not be admissible.

Database	Optimality Known?	Length of Optimal Tour	n =
Barachet	Yes	948	10
Preparata	Yes	1774	16
Loop	No	1919	20
E-figure	No	1258	37
S-figure	No	1770	41
Capitals	Yes	3352	48
Augsburg	Yes	4731	127

Figure 1. The seven benchmark databases, with associated optimality information. The tour lengths are expressed as a function of pixels of the computer raster.

An Analysis of the Barachet Dataset.

Figure 2 is a visual graphic of a shortest tour evolving in time as interior cities are incrementally processed. The data is the Barachet dataset, published in 1957 [B1]. The original constellation of cities is shown in the upper left corner, followed by a graphic of the convex hull, which in this case is simply a square. The dynamic programming algorithm then proceeds to add each of the five interior cities to the tour. The simple extensions, represented by the H_E operator, turn out to be not very interesting. The insertions which produce the most profound changes are the interchange operators, designated H_S. As an example, the last state (frame nine) is produced by an interchange. The extension shown in the next-to-last frame causes the upper perturbation to yield two cities to the extended perturbation as at frame nine, while at the same time producing a new perturbation from the top, which was seen once before at frame number three. Of course, the sequence would look quite different if the interior cities were to be inserted in an order other than that dictated by the process of nested hull traversal, but the final tour would look the same.

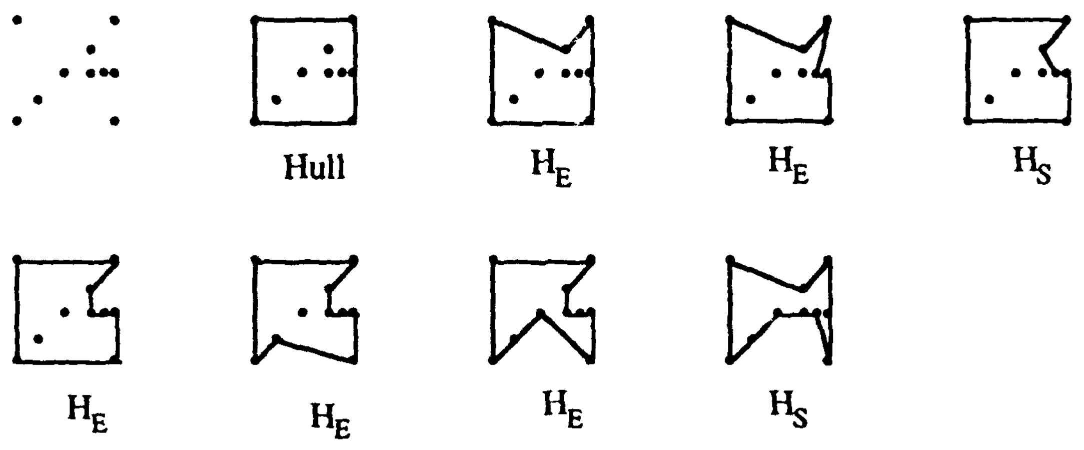


Figure 2. Incremental Optimality Portrayed for the Barachet Data

An Analysis of the Loop Dataset.

Figure 3 is a tabular description of the algorithmic logic manifested when processing the twenty city loop figure (a graphic temporal history of the logic is contained in the appendix). Although the extension and shunting operations are well represented, there are no cross-hull interchanges which occur in this database. The deferral operations occur because the city under consideration forms a more acute angle with the current tour than does some other sample in the queue of interior cities. In such cases, the city forming the more acute angle is placed back in the queue, and the other city is brought forward for processing. Actually, the deferral and extension operations are mundane when compared to shunts and interchanges. The two interesting insertions for this dataset are the shunts introduced by the addition of cities 19 and 112, both of which radically alter the global tour shape. In particular, the insertion of city 112 causes the lower right portion of the tour to change from a "fishtail" shape to a concave loop. It should be emphasized that the insertion of the cities in some other order might cause the ultimate loop behavior to be displayed earlier, but the algorithm is designed to display the shortest tour for only the cities which are currently entered. A partial tour for k cities may or may not structurally resemble the shortest tour for all n cities.

Entered City	Insertion Operation	Relevant Subtour
15	deferral	-
119	extension	(4,19,20)
15	extension	(4,5,19,20)
116	extension	(20,16,2)
115	extension	(20,16,15,2)
114	extension	(20,16,15,14,2)
113	deferral	-
112	deferral	-
117	extension	(20,17,16,15,14,2)
113	extension	(20,17,16,15,14,13,2)
112	deferral	-
118	extension	(20,18,17,16,15,14,13,2)
112	deferral	-
17	extension	(4,7,5,19,20)
112	deferral	-
16	extension	(4,7,6,5,19,20)
112	deferral	-
18	extension	(4,8,7,6,5,19,20)
112	deferral	-
19	leftsided shunt	(3,9,8,7,6,4,5,19,20)
112	leftsided shunt	(4,5,6,7,8,9,12,13, 14,15,16,17,18,19,20)
110	extension	(4,5,6,7,8,9,10,12,13, 14,15,16,17,18,19,20)
111	extension	(4,5,6,7,8,9,10,11,12,13, 14,15,16,17,18,19,20)

Figure 3. The dynamic programming result for the twenty city loop figure. The insertions of cities 19 and 112 produce back-to-back shunting operations, each of which radically alters the visual appearance of the optimal tour. A temporal history of this example (the loop dataset) is contained in the appendix.

An Analysis of the United States Capitals Dataset.

The appendix concludes with two graphics which depict the development of the shortest tour for the forty-eight capitals of the contiguous United States. The first graphic demonstrates the same approach described above for the Barachet and loop datasets: i.e., a baseline tour consisting of the outer hull is established, and and the interior cities are inserted incrementally by probing inward one hull at a time until all cities are exhausted. The first interesting behavior occurs at row three, column five, with the introduction of Little Rock: Oklahoma City and Jackson are interchanged into Little Rock's new perturbation. Another interchange occurs at row four, column two, when Frankfort is extended into Charleston's perturbation, which subsequently causes Montgomery to be interchanged from below. Yet another interchange occurs in row four, column six, when the introduction of Cheyenne first causes extension space to transpose Bismarck with Pierre, and then forces the interchange of Salt Lake City. The final interchange occurs in row five, column three, when the newly introduced city of Lansing compels the cities of Albany and Harrisburg to be absorbed into Lansing's perturbation. By far, the most dramatic behavior is encountered at row six, column seven, when the introduction of Springfield forces a left shunt. Springfield is originally attached by extension space between Nashville and Frankfort, but the shunt operator then links it to Jefferson City and synthesizes a new perturbation issuing from the hull segment with endpoints consisting of Baton Rouge and Tallahassee. The final tour shown in row six, column 2 was proven optimal by AT&T Bell Laboratories in 1985 [A1].

Turning to the second graphic in the appendix concerning the forty-eight capitals, the alternative convex hull approach is utilized. This time, the initial tour consists of the innermost hull in the nested decomposition, with vertices comprised of Des Moines, Springfield, Indianapolis, and Columbus. The nested hulls exterior to this hull are subsequently processed, beginning with the one nearest to the inner hull. Because the quartic and hyperbolic loci remain valid regardless of the processing order, the same optimal tour is ultimately obtained at row six, column two.

Some Remarks about the Augsburg Dataset.

The Augsburg dataset consists of the locations of one hundred twenty-seven beer gardens in the city of Augsburg, Germany. This dataset has been solved to optimality by German researchers at the University of Augsburg, using a variant of branch-and-cut [R1]. The same optimal tour is obtained by the dynamic programming algorithm described in this paper. However, analysis of this dataset will not be described here, since it will serve as the primary example in the development of a theorem to be published in a forthcoming paper [C3].

Scaling Up: a 532-City Dataset.

A five hundred thirty-two city dataset was developed by Shen Lin when he was employed at AT&T Bell Laboratories, and represents the locations of AT&T telephone offices in the contiguous United States. A certificate of optimality has been obtained for this data by the originators of the branch-and-cut algorithm [P1]. This database is intriguing because it is the largest database certified to date for which the cities are randomly positioned in the plane (a 2392-city dataset has been solved, but the constellation of cities is formed by repeating the same small pattern of cities several times). The dynamic programming algorithm has not yet been brought to bear upon this database, but it may be feasible to describe the result of its application in the same paper in which the one hundred twenty-seven city solution is discussed.

Time Complexity: A Worst-case Analysis.

The dynamic programming algorithm is continuing to evolve as a research and development tool, and as such remains suboptimal. Nevertheless, it is instructive to perform a worst-case analysis of the code as currently implemented. A condensed algorithmic flowchart is shown at Figure 4. The label "In" is the input loop, in which a new city is input from the front of the queue of unprocessed interior cities. Upon entry from the queue, the city is processed by a routine which checks for intra-perturbation optimality. The new city is first compared against every segment in the current tour to discover the segment of least elliptic distance. This segment may or may not contain the new city's nearest neighbor, so a subroutine is called to check the tour length if an alternative hypothesis allows the connection to occur. To encourage the gradual introduction of cities relative to the perturbed hull, if some other interior city forms a larger angle with the current tour, it is brought forward for processing and the candidate city is put on hold. The intra-perturbation routine concludes by reordering the city's perturbation if necessary to achieve optimality.

Next, a global heuristic is applied to determine if some tour segment forms a larger angle with the newly inserted city than the segment to which it was attached locally via the elliptic distance computation. The global interchange operator attaches the city to such a segment if it exists, and triggers a quadratic matching operation in an attempt to absorb cities from other perturbations. Next in the processing sequence is the synthesis of the left and right shunt topologies. The extended perturbation is attached to both the nearest perturbation on the left and the nearest perturbation on the right, and new perturbations are generated respectively to the left and the right, between the perturbations maintained by the extension space. Once the shunts are computed, the tour lengths for the extension space, the left shunt space, and the right shunt space are compared, and the minimal topology is preserved.

At this point, the interchange operator is invoked once again to absorb cities from other perturbations, using the left and right extension edges of the new city's position as a baseline perturbation. To wrap up the processing of the city, some housekeeping operations are performed to commit the tour and its length to computer memory, before returning to the input loop to process any remaining interior cities.

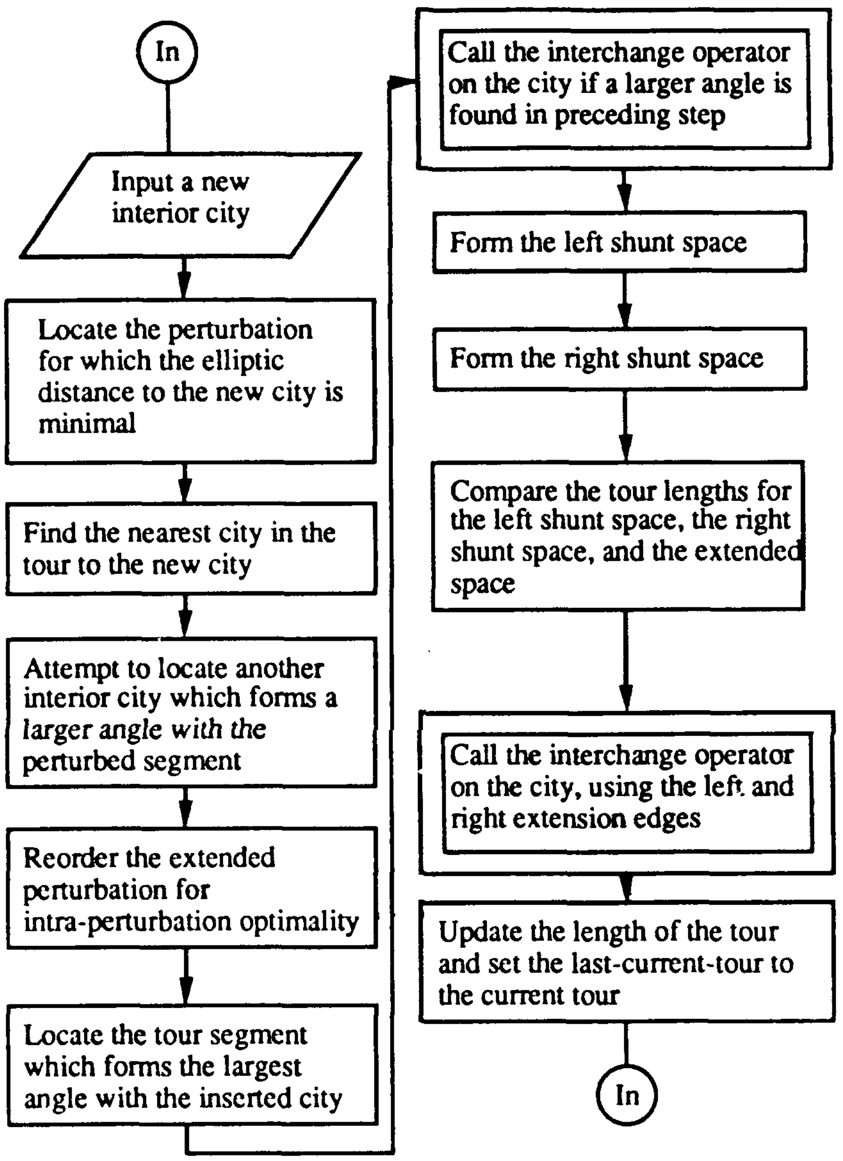

In
Input a new interior city
Locate the perturbation for which the elliptic distance to the new city is minimal
Find the nearest city in the tour to the new city
Attempt to locate another interior city which forms a larger angle with the perturbed segment
Reorder the extended perturbation for intra-perturbation optimality
Locate the tour segment which forms the largest angle with the inserted city
Call the interchange operator on the city if a larger angle is found in preceding step
Form the left shunt space
Form the right shunt space
Compare the tour lengths for the left shunt space, the right shunt space, and the extended space
Call the interchange operator on the city, using the left and right extension edges
Update the length of the tour and set the last-current-tour to the current tour
In

Figure 4. A high level flowchart of the dynamic programming algorithm as currently implemented. For the sake of brevity, several comparison operations of complexity O[1] have been omitted. An interchange operation (double box) is relatively expensive; each interchange entails a quadratic matching, four sorts, and four linear searches.

As the k^{th} city is processed, it is possible for it to trigger two searches of quadratic complexity, eight sorts, each of complexity $k*\log k$, and fifteen searches of linear complexity. All of the boxes in the flowchart represent processes of linear complexity or faster, except for the two double boxes representing the interchange processes. The interchanges are more expensive, in that they involve quadratic matching and sorting in a quest to globally merge perturbations which may lie on opposite sides of the hull. Finally, there is a constant overhead ζ_{OS} associated with the computer operating system and hardware suite. Therefore, the worst-case processing time t_w is bounded by the following cubic expression:

$$\begin{aligned} t_w &= \Sigma\ [\, 2\,k^2 + 8\,k * \log k + 15\,k + \zeta_{os} \,] \\ &\leq 2/6\, n * (n+1) * (2n+1) + 8/2\, n * (n+1) * \log n + 1/2\, n * (n+1) + \zeta_{os}\, n \\ &= 2\,n^3 + 4\,n^2 * \log n + 7/2\, n^2 + 4\,n * \log n + (3/2 + \zeta_{os})\, n\,. \end{aligned}$$

Complexity theorists refer to such a bound as a ceiling function, because it is derived empirically from an algorithm which has not yet been proven to be optimal, and in general must be considered inferior to a theoretical bound on performance. Conversely, a floor function is obtained analytically from worst-case analysis of an algorithm known to terminate with an optimal solution (usually, once a floor function is established by theory, progress is rapid in bringing an algorithmic ceiling function down to converge upon the floor function). Since a floor function has to date not been established for the Euclidean traveling salesman problem, it is necessary to attempt to empirically lower the ceiling function by resorting to heuristic techniques. The operators depicted in the flowchart are heuristic techniques designed to model the non-linear search space reported upon at [C2]. The intrapath operators at the left represent the hyperbolic portion of the locus, while those on the right approximate the discriminator for the fourth-order components. Many of the minor processing steps which are of sublinear complexity are intentionally omitted, to afford the reader as concise a view as possible of the global logic. The author wishes to stress that the implementation is at best a stopgap measure, which is a useful research tool only until more geometric facts about the search space become available. Indeed, the suggested implementation is already obsolete, due to a new theorem with the potential to dispatch a significant portion of the interior cities during a fast preprocessing step [C3].

An Experiment to Test the Validity of the Analytic Cubic Bound.

Tables 1-8 on the next page are a compilation of a set of experiments designed to test the validity of the cubic bound developed in the preceding section of the paper. The algorithm as currently implemented was tasked against sets of cities randomly distributed on a computer screen (the author used the computer mouse to rapidly input a set of random points to the screen, which were then utilized as coordinates for a city database). The number of cities simulated was allowed to vary from ten to forty-five, in increments of five. For a specific number of cities n, ten sets of random data of size n were generated. Each set was processed by the dynamic programming algorithm, and the following parameters were monitored by the computer operating system: *space* (the number of Lisp cons cells, or computer words, consumed by the run); *time* (the number of seconds of central processing unit time consumed by the run); and *allocation* (the number of seconds of CPU time devoted to dynamic reclaiming of memory, using the Lisp garbage collector).

Only the CPU time (the central column of each dataset) was analyzed statistically. The sample mean, variance, and standard deviation were computed for each set of CPU time data. In addition the best and worst run time outliers were selected for each set. It was anticipated that the worst case outlier would be a good candidate to compare against the cubic bound predicted by the analysis.

Figure 5 is a line graph of the experimental results. For the eighty runs of the algorithm listed in the appendix, the best-case, average-case, and worst-case running times are plotted for each of the eight groupings of ten cities. Also included in the same plot is the cubic bound; the bound is computed for each value of n, and is scaled by the constant .0075 to render the graphic more compact in the ordinate dimension.

It is perhaps imprudent to extrapolate for values of n larger than those shown, but the cubic bound predicted by the theory appears to be a reasonable ceiling function for the worst-case performance of the algorithm. Although there is a gap between the bound and the worst-case outlier, there are valid explanations. One explanation is that an insufficient number of samples were selected to see true worst-case behavior. Another explanation is that the author was overly conservative when conducting a worst-case analysis of the computer code, causing the cubic bound to be somewhat inflated. Yet another explanation is that a more judicious selection of a scalar multiplier of the cubic expression could close the gap. The important thing to note is that the bound is visually well-correlated with the worst-case plot, and that the general behavior of the two curves is markedly similar.

	Space	Time	Allocation
1	7337	12.599	3.380
2	5943	9.739	3.689
3	3271	5.717	1.480
4	2772	4.879	0.000
5	2539	4.342	1.267
6	4726	8.155	2.326
7	3641	6.437	1.579
8	3314	5.714	1.630
9	3245	6.021	1.427
10	4192	7.116	1.399

$n = 10$, $\overline{X} = 7.712$, $S^2 = 10.130$, $S = 3.183$

	Space	Time	Allocation
1	10841	16.515	5.122
2	11926	16.248	8.381
3	13918	20.285	7.593
4	13347	18.926	8.388
5	17042	23.476	9.775
6	15003	20.873	9.793
7	9214	13.009	5.688
8	20864	31.220	11.403
9	9755	13.709	6.302
10	13369	19.800	7.806

$n = 15$, $\overline{X} = 19.406$, $S^2 = 27.906$, $S = 5.283$

	Space	Time	Allocation
1	31893	39.953	18.919
2	39848	52.327	23.654
3	31961	41.991	19.001
4	33496	44.131	19.732
5	28844	36.980	18.365
6	41600	50.265	28.959
7	27420	36.007	17.767
8	27191	34.494	17.022
9	29502	37.315	17.940
10	28340	35.103	18.456

$n = 20$, $\overline{X} = 40.857$, $S^2 = 39.732$, $S = 6.303$

	Space	Time	Allocation
1	63375	72.603	38.143
2	50002	47.761	41.782
3	48373	56.510	27.231
4	52179	67.577	30.072
5	53730	65.259	32.730
6	63618	77.347	39.047
7	67187	81.024	40.680
8	64343	78.275	37.596
9	62891	75.275	35.368
10	49037	56.832	28.396

$n = 25$, $\overline{X} = 67.846$, $S^2 = 123.389$, $S = 11.108$

	Space	Time	Allocation
1	102662	121.708	58.073
2	101337	125.221	61.622
3	92058	98.096	66.107
4	108662	132.876	64.330
5	104230	129.015	62.486
6	108849	134.460	66.508
7	82841	97.734	46.839
8	103552	68.177	119.213
9	84362	133.353	285.494
10	101140	124.738	59.313

$n = 30$, $\overline{X} = 116538$, $S^2 = 469.822$, $S = 21.675$

	Space	Time	Allocation
1	150861	185.943	101.698
2	121924	152.557	86.844
3	161438	197.643	114.430
4	186448	246.866	129.780
5	163579	204.787	114.469
6	161428	212.644	112.123
7	178192	227.529	121.439
8	214286	295.458	152.270
9	180710	196.557	105.282
10	156222	167.035	64.046

$n = 35$, $\overline{X} = 208.702$, $S^2 = 1671.621$, $S = 40.885$

	Space	Time	Allocation
1	209331	224.835	98.215
2	241452	267.132	99.419
3	206614	212.946	85.847
4	263413	295.444	115.531
5	295821	348.702	136.283
6	239564	257.743	110.176
7	229468	263.191	106.129
8	223627	239.093	97.718
9	216796	249.739	96.802
10	210261	244.238	91.558

$n = 40$, $\overline{X} = 260.306$, $S^2 = 1492.335$, $S = 38.631$

	Space	Time	Allocation
1	329234	389.722	244.246
2	354376	411.498	156.177
3	382554	490.126	179.530
4	426650	593.475	208.267
5	293465	301.532	114.221
6	404882	505.157	173.208
7	384530	530.231	278.536
8	341033	413.296	213.390
9	335337	389.147	196.894
10	278591	344.297	178.995

$n = 45$, $\overline{X} = 436.848$, $S^2 = 8147.415$, $S = 90.263$

Tables 1-8. Space, Time, and Allocation Complexity for Eight Sets of ETSP Experiments

(*n* is the number of cities per experiment; *Space* is the number of Lisp cons cells consumed by a run; *Time* is the number of seconds of CPU time consumed by a run; and *Allocation* is the time dedicated to the Lisp garbage collector)

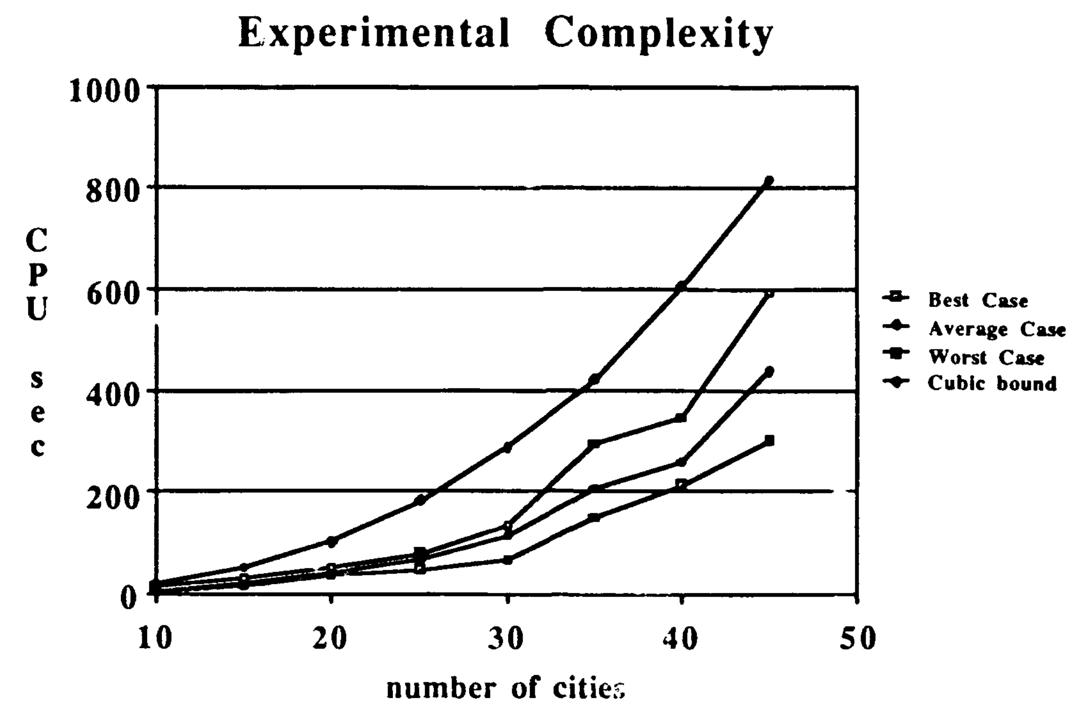


Figure 5. A line plot of the time complexity of the dynamic programming algorithm as a function of the number of cities processed.

Summary.

Some very preliminary statistical experiments on the worst case behavior of an algorithm designed to provide an exact solution for the Euclidean traveling salesman problem indicate that the run time of the algorithm is in fact bounded by an expression cubic in the number of cities. The algorithm is based on some recent theoretical results pertaining to the non-linear search space for the Euclidean traveling salesman problem, and as such the computer code has not yet reached an optimal level of maturity. Nevertheless, it has proven useful to statistically compare the performance of the dynamic programming algorithm against the cubic bound predicted by a cursory examination of the currently implemented software. The worst-case statistical outliers compiled for each set of experiments are indeed bounded by the cubic expression developed analytically from the current formal design specification of the algorithm. It is apparent that the science of statistics is invaluable with regard to gauging the probabilistic performance of an algorithm versus its analytic time complexity bound.

Future Directions of the Research.

An entirely independent issue is whether or not the algorithm is admissible: i.e., whether it terminates with an optimal solution It is desirable to attempt to prove that the dynamic programming technique is admissible; a proof by induction seems promising. Thus far, the implementation is proceeding in the spirit of the Hungarian mathematician Lakatos, who contended that a theory is never truly proven until sufficient time has passed such that the community at large accepts the theory, based on the fact that counterexamples cease to be forthcoming from empirical testing [L1]. The implementation discussed in the paper is at a stage where counterexamples can still be found. However, the author feels that the counterexamples are sufficiently trivial to be local rather than global, which indicates that the problems remaining to be ironed out are details of implementation rather than profound issues of conceptualization. It seems important to pursue the proof of optimality; otherwise, the new algorithm will be vulnerable to the same kinds of criticism which plague all heuristic approaches to problem solving.

The algorithm will continue to undergo empirical testing, as the number of cities is scaled up. A good source of benchmarks is maintained at reference [R1]. As mentioned above, a one hundred twenty-seven city database has recently been solved to optimality by the dynamic programming algorithm. It is desirable in 1991 to move ahead to a five hundred thirty-two city certified benchmark [P1]. Unfortunately, there are only a handful of large databases for which a certificate of optimality has been obtained.

In preparation is a paper which describes a new geometric result pertaining to the aspect angle which an interior city forms with the convex hull, and the positive implication of the result as a preprocessing step for the dynamic programming algorithm [C3]. It is premature to forecast the utility of the new theorem, but empirical testing indicates that on the average a surprisingly large percentage of cities interior to the hull may be inserted into the tour in a fast preprocessing step.

Acknowledgments

The research has benefited from technical discussions with the following individuals: Gerald Andersen, Richard Antony, Robert Bixby, Chris Bogart, Paul Broome, Mel Brown, Jagdish Chandra, Douglas Chubb, Ben Cummings, Michael Dillencourt, Francis Dressel, Herbert Edelsbrunner, Geoffrey Fox, Ray Freeman, Martin Groetschel, Andrew Harrell, Robert Hein, William Jackson, David S. Johnson, Shen Lin, Sanjoy Mitter, Jay Sethuraman, J. Michael Steele, Andrew Thompson, Paul Tseng, Franz-Erich Volter, and David Willshaw. I am deeply indebted to Richard Antony for his comprehensive technical review, requests for elaboration, and patient editing of the final draft. A special note of gratitude to Geoffrey Fox and Robert Bixby for an invitation to a productive workshop on computational aspects of the traveling salesman problem (held at Rice University in April of 1990); to David S. Johnson for constructive criticism of the dynamic programming algorithm, and for providing the coordinates for the forty-eight capitals; and to Herbert Edelsbrunner for insights into computing the convex hull. I also wish to thank the management of the CECOM Center for Signals Warfare for establishing a scientific environment in which fundamental research in mathematics and computer science is both encouraged and supported.

Bibliography

[A1] Anonymous, "Here's one for the road, and then some", *Discover*, July 1985.

[B1] Barachet, L.L., "Graphic Solution of the Traveling Salesman Problem", **Operations Research 5**, 1957, pp. 841-845.

[B2] Bellmore, M., and G.M. Nemhauser, "The Traveling Salesman Problem: a Survey", **Operations Research 16**, 1968, pp. 538-558.

[C1] Chazelle, B., "On the convex layers of a convex set", **IEEE Trans. Info. Theory IT-31**, 1985.

[C2] Cronin, T.M., "The Voronoi Diagram for the Euclidean Traveling Salesman Problem is Piecemeal Quartic and Hyperbolic", *Transactions of the Eighth Annual Army Conference on Applied Mathematics and Computing*, Cornell University, June 1990.

[C3] Cronin, T.M., "A Note on the Aspect Angle formed between the Convex Hull and its Interior Points, within the Context of Shortest Euclidean Tours", in preparation.

[F1] Flood, M.M., "The Traveling Salesman Problem", **Operations Research 4**, 1956, pp. 61-75.

[G1] Garey, M.R., R.L. Graham, and D.S. Johnson, "Some NP-complete Geometric Problems", Eighth Annual Symp. on Theory of Comp., May 1976, pp. 10-22.

[G2] Garey, M.R., and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, New York NY, 1979.

[H1] Held, M., and R.M. Karp, "The Traveling Salesman Problem and Minimum Spanning Trees: Part II", **Mathematical Programming 1**, 1971, pp. 6-25.

[J1] Johnson, D.S., Private communication, and set of viewgraphs entitled "How to Beat Lin-Kernighan", *Workshop on Computational Aspects of the Traveling Salesman Problem*, Rice University, Houston TX, April 1990.

[J2] Johnson, D.S., "Local Optimization and the Traveling Salesman Problem", to appear in the *Proceedings of the Seventeenth Colloquium on Automata, Languages, and Programming*, Springer-Verlag, 1990, pp. 446-461.

[L1] Lakatos, I., Proofs and Refutations: The Logic of Mathematical Discovery, Cambridge University Press, New York NY, 1976.

[L2] Lawler, E.L., J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, eds., The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, John Wiley and Sons, New York NY, 1985.

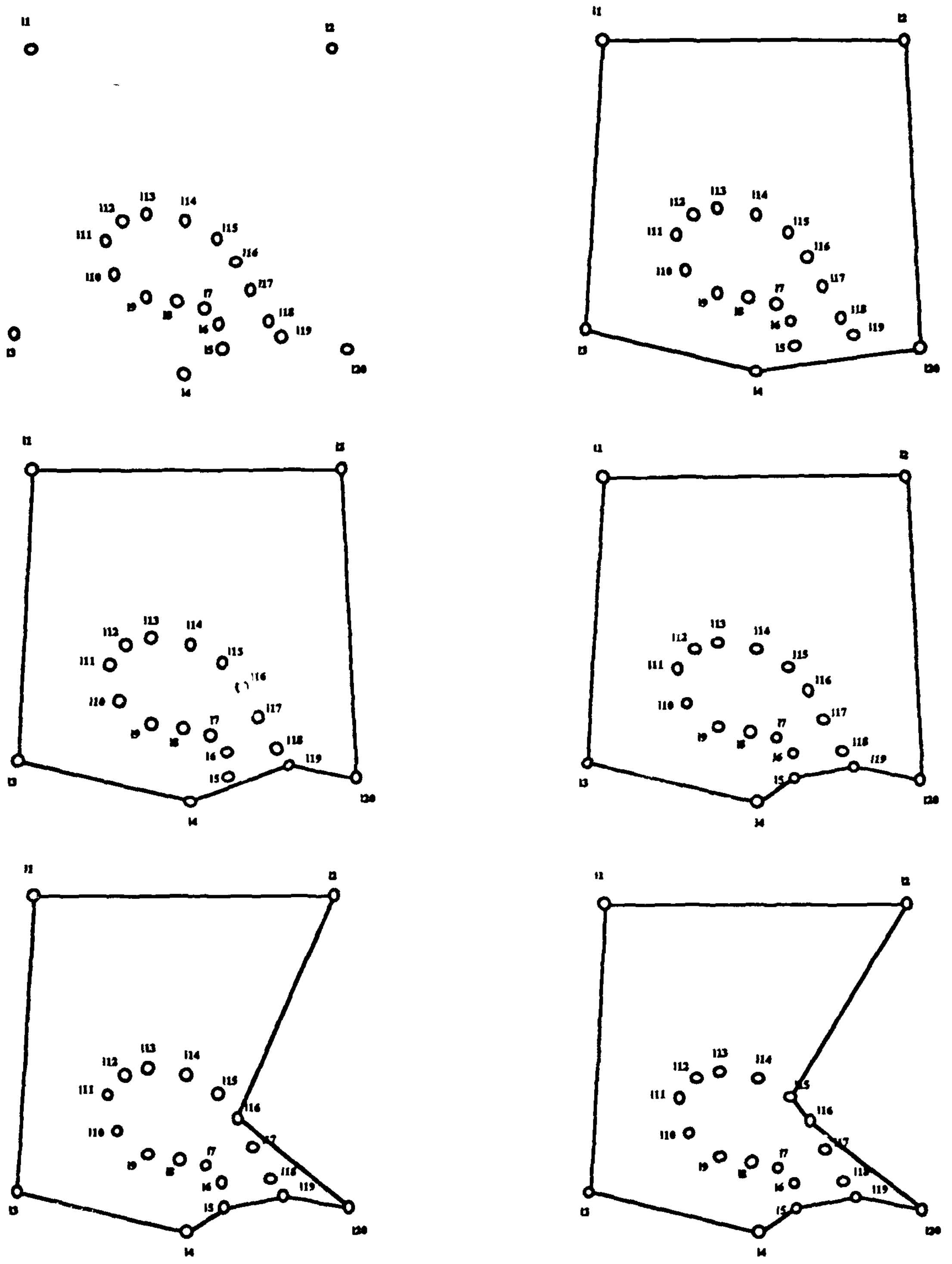
[L3] Lin, S., and B.W. Kernighan, "An effective heuristic algorithm for the traveling salesman problem", **Operations Research 21**, 1973, pp. 498-516.

[P1] Padberg, M., and G. Rinaldi, "Optimization of a 532-city Symmetric Traveling Salesman Problem by Branch and Cut", **Op. Res. Letters, Vol 6, No.1**, March 1987.

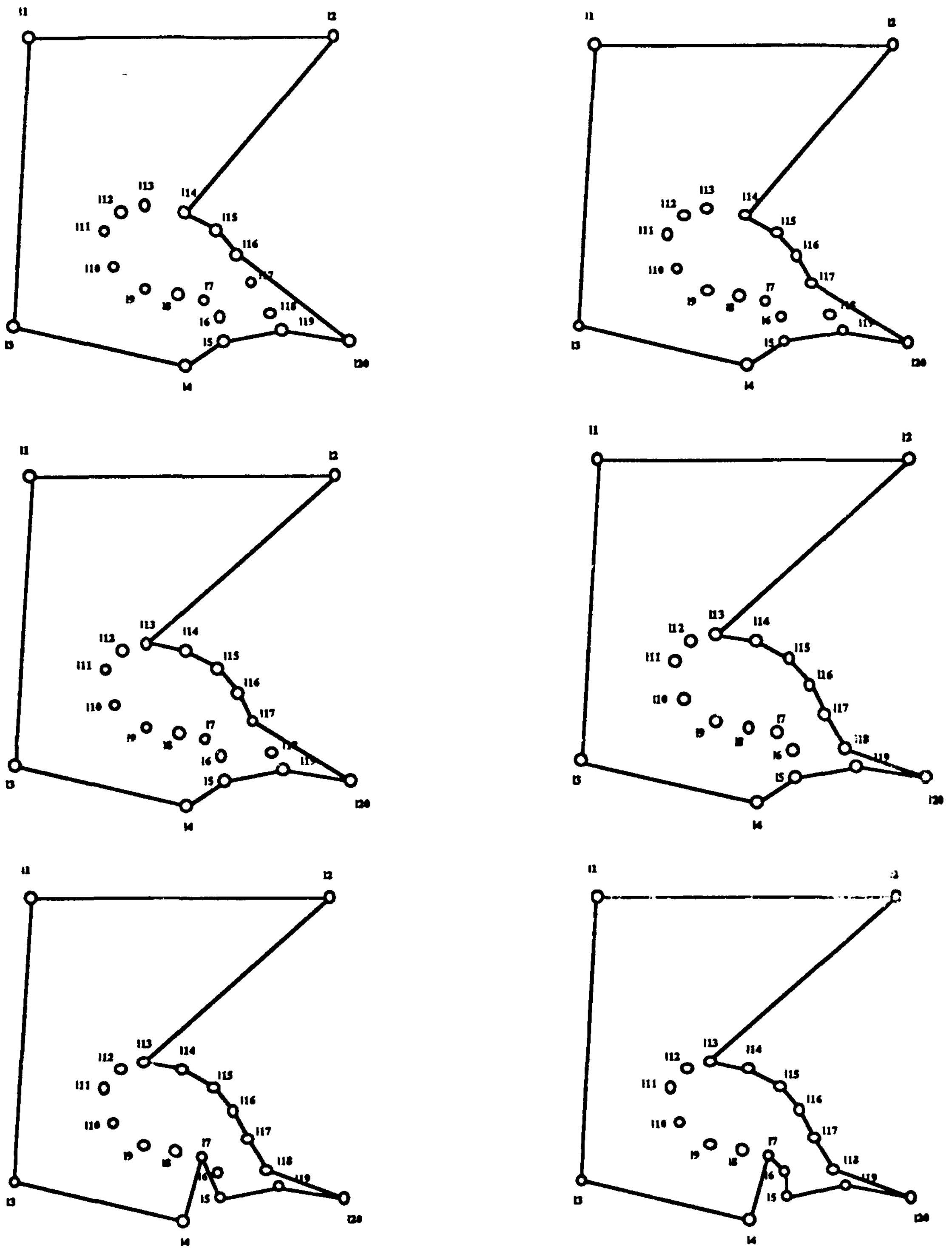
[P2] Preparata, F.P., and M.I. Shamos, Computational Geometry: An Introduction, Springer-Verlag, New York NY, 1985.

[R1] Reinelt, G., "TSPLIB - A Traveling Salesman Problem Library", Institute of Mathematics, University of Augsburg, Augsburg Germany, 20 March 1989.

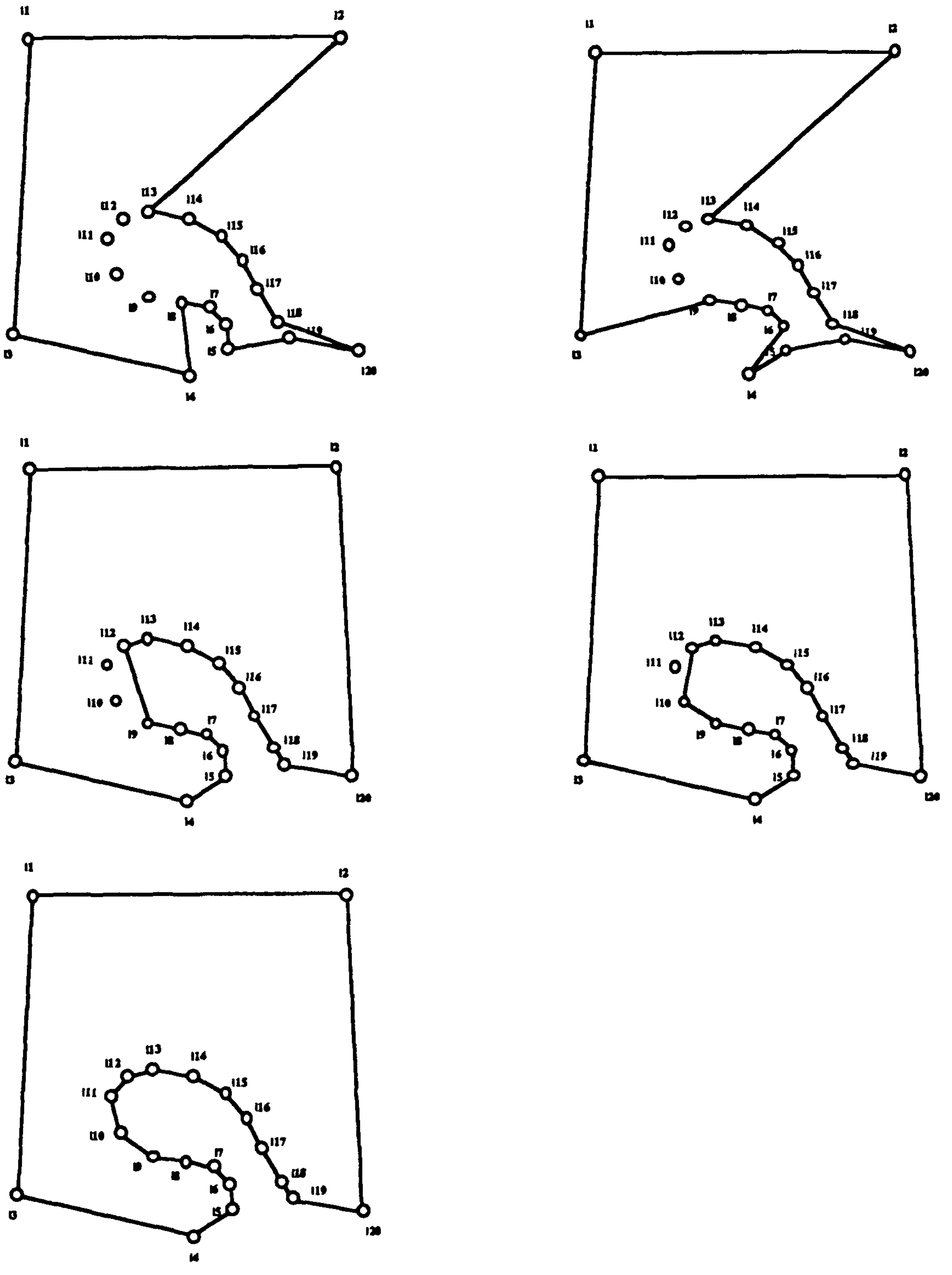
[W1] Walpole, R.E., and R.H. Myers, Probability and Statistics for Engineers and Scientists, Macmillan Publishing Company, Inc., New York NY, 1972.



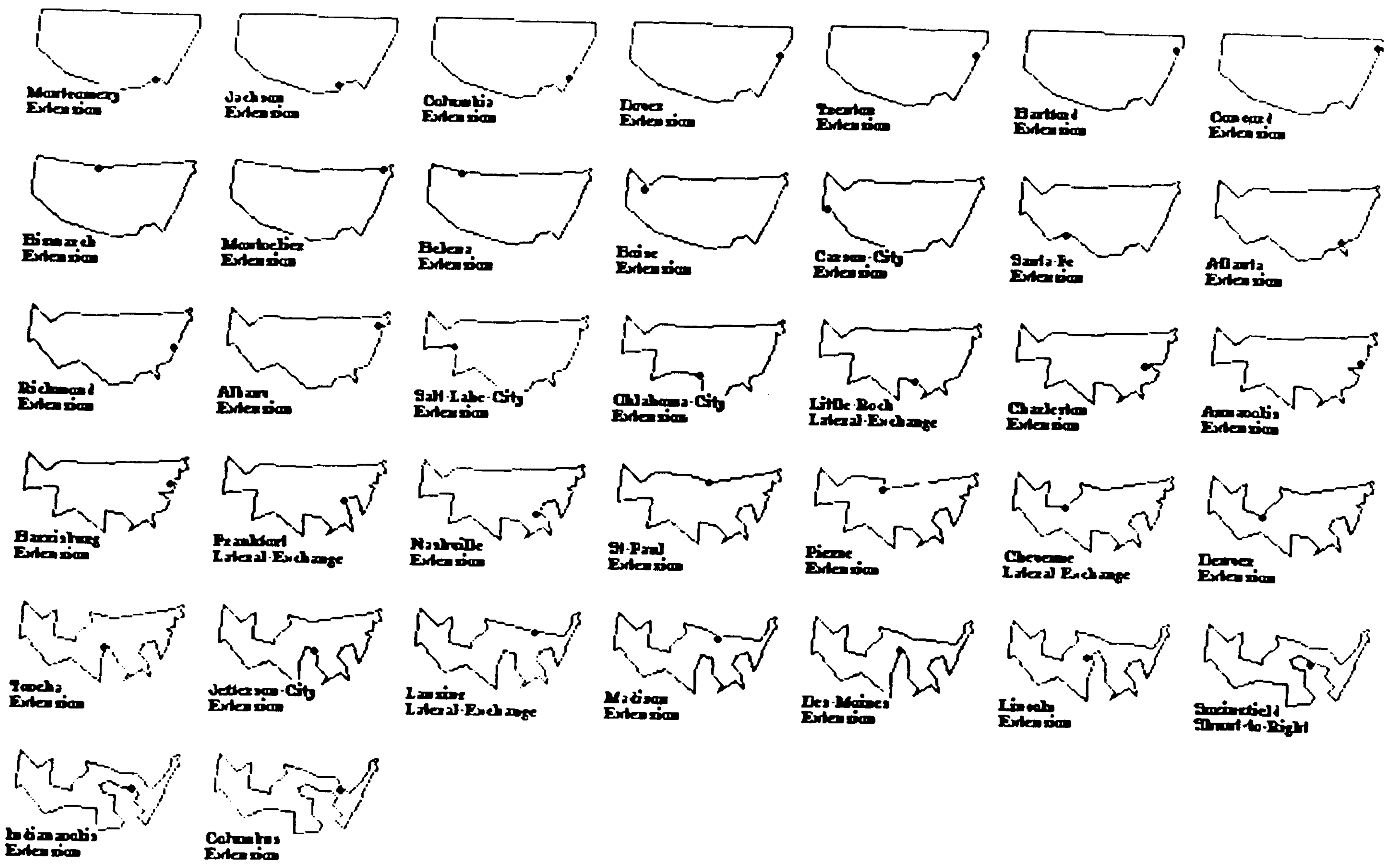
Loop dataset: original data; convex hull;
dynamic programming insertion of interior cities l19; l5; l16; l15.

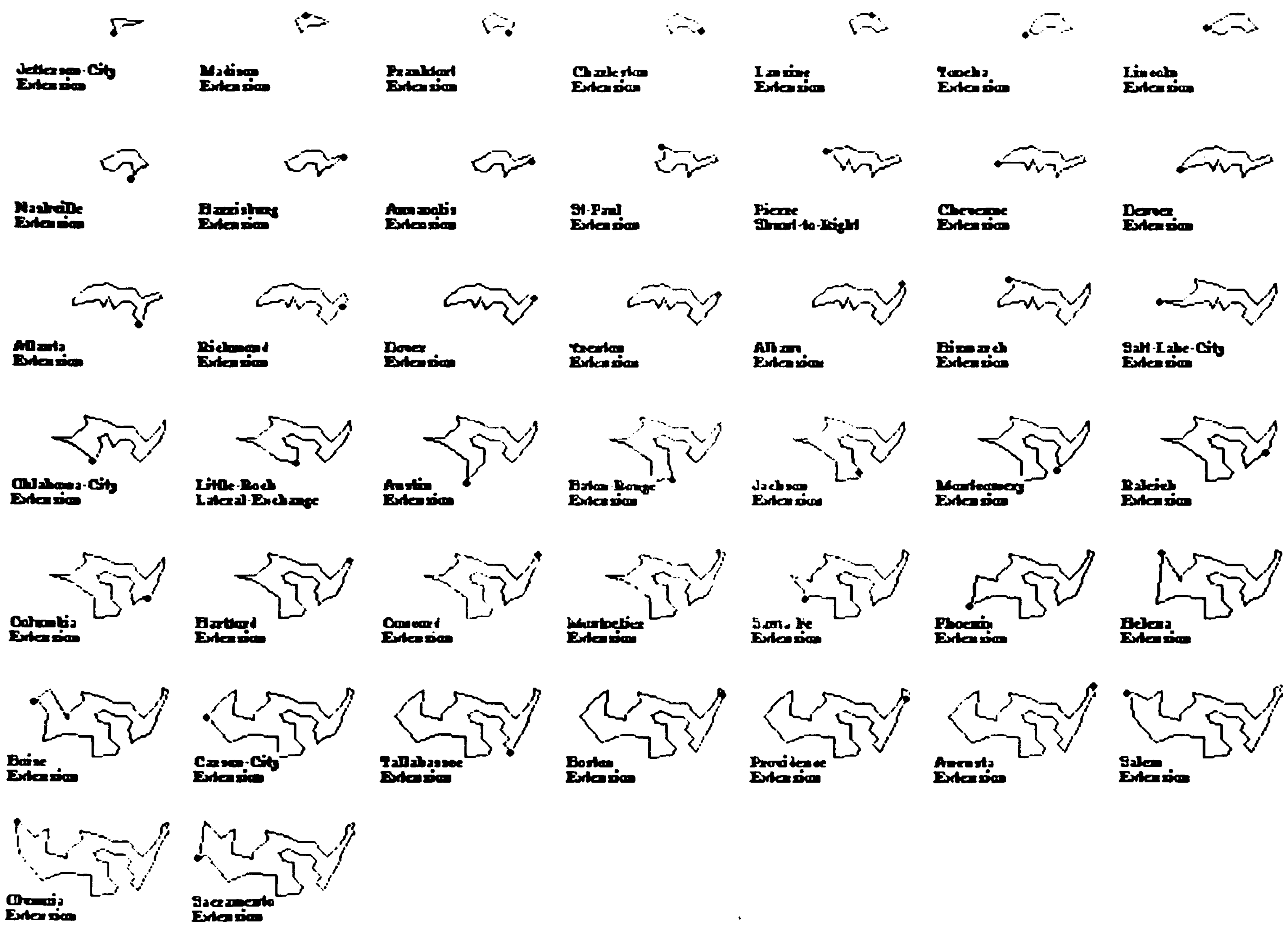


Loop dataset: dynamic programming insertion of interior cities l14; l17; l13; l18; l7; l6.



Loop dataset: dynamic programming insertion of interior cities l8; l9; l12; l10; l11.


Montgomery Extension
Jackson Extension
Columbia Extension
Dover Extension
Trenton Extension
Hartford Extension
Concord Extension
Bismarck Extension
Montpelier Extension
Helena Extension
Boise Extension
Carson City Extension
Santa Fe Extension
Atlanta Extension
Richmond Extension
Albany Extension
Salt Lake City Extension
Oklahoma City Extension
Little Rock Lateral Exchange
Charleston Extension
Annapolis Extension
Harrisburg Extension
Frankfort Lateral Exchange
Nashville Extension
St Paul Extension
Pierre Extension
Cheyenne Lateral Exchange
Denver Extension
Topeka Extension
Jefferson City Extension
Lansing Lateral Exchange
Madison Extension
Des Moines Extension
Lincoln Extension
Springfield Shunt-to-Right
Indianapolis Extension
Columbus Extension


Jefferson-City Extension
Madison Extension
Frankfort Extension
Charleston Extension
Lansing Extension
Topeka Extension
Lincoln Extension
Nashville Extension
Harrisburg Extension
Annapolis Extension
St-Paul Extension
Pierre Short-to-Right
Cheyenne Extension
Denver Extension
Atlanta Extension
Richmond Extension
Dover Extension
Trenton Extension
Albany Extension
Bismarck Extension
Salt-Lake-City Extension
Oklahoma-City Extension
Little-Rock Lateral-Exchange
Austin Extension
Baton-Rouge Extension
Jackson Extension
Montgomery Extension
Raleigh Extension
Columbia Extension
Hartford Extension
Concord Extension
Montpelier Extension
Santa Fe Extension
Phoenix Extension
Helena Extension
Boise Extension
Carson-City Extension
Tallahassee Extension
Boston Extension
Providence Extension
Augusta Extension
Salem Extension
Olympia Extension
Sacramento Extension

]